

O propósito deste trabalho é mostrar as diferenças no comportamento de semânticas de algumas construções comumente empregadas em linguagens de programação por meio da implementação dessas construções em uma máquina de pilha.

A máquina de pilha, disponibilizada em conjunto com este enunciado, e disponível em <https://github.com/lives-group/StackVM>, possui uma memória que é um arranjo de valores começando no índice e por padrão tem 4096 células de memória. Cada célula pode armazenar um valor independentemente do tamanho do valor. Valores podem ser inteiros, pontos flutuantes, booleanos e caracteres. Para efeitos de simplicidade de implementação caracteres são modelos como valores inteiros.

Como usual a máquina contém uma memória acessada com a política de pilha, separada da memória principal e potencialmente infinita. Somente se pode realizar operações sobre valores que estão na pilha.

Um programa é formado por uma sequência de instruções mnemônicas, que são descritas nas tabelas 1 e 2 a seguir. A fim de simplificar o texto que explica o funcionamento de cada instruções, usa-se a seguinte notação para referenciar elementos na pilha:  $x_n \dots x_0$ , onde  $x_n$  representa o topo da pilha e  $x_0$  representa o elemento na base da pilha.

Instruções Aritméticas - Inteiros e Ponto flutuante	
<b>addi, addf</b>	desempilha $x_n$ e $x_{n-1}$ e empilha $x_n + x_{n-1}$
<b>subi, subf</b>	desempilha $x_n$ e $x_{n-1}$ e empilha $x_n - x_{n-1}$
<b>multi, subf</b>	desempilha $x_n$ e $x_{n-1}$ e empilha $x_n * x_{n-1}$
<b>divi, divf</b>	desempilha $x_n$ e $x_{n-1}$ e empilha $x_n / x_{n-1}$
Instruções Lógicas e de Comparação	
<b>and</b>	desempilha $x_n$ e $x_{n-1}$ e empilha $x_n \wedge x_{n-1}$
<b>or</b>	desempilha $x_n$ e $x_{n-1}$ e empilha $x_n \vee x_{n-1}$
<b>not</b>	desempilha $x_n$ e empilha $\neg x_n$
<b>lt</b>	desempilha $x_n$ e $x_{n-1}$ e empilha $x_n < x_{n-1}$
<b>gt</b>	desempilha $x_n$ e $x_{n-1}$ e empilha $x_n > x_{n-1}$
<b>eq</b>	desempilha $x_n$ e $x_{n-1}$ e empilha $x_n == x_{n-1}$

Tabela 1: Conjunto de instruções para manipulação de floats, inteiros e booleanos da máquina de pilha

Nas instruções da Tabela 2, a seguir, o acesso a memória é denotado por  $m[i]$  que significa o valor na  $i$ -ésima posição da memória.

Instruções de Controle de Memória	
<b>load</b>	desempilha $x_n$ e empilha $m[x_n]$
<b>store</b>	desempilha $x_n$ e faz $m[x_n] = x_{n-1}$ . O valor $x_{n-1}$ é mantido na pilha
<b>push &lt;n&gt;</b>	empilha $n$
<b>pop</b>	desempilha $x_n$
<b>dup</b>	duplica o valor de $x_n$ .
Instruções de controle de Fluxo de Execução	
<b>jump &lt;j&gt;</b>	Salta a instrução na $j$ -ésima posição
<b>jump &lt;j&gt;</b>	desempilha $x_n$ e se $x_n = true$ salta para a $j$ -ésima instrução.
<b>jumps</b>	desempilha $x_n$ e salta para $x_n$ -ésima instrução
<b>jumpst</b>	desempilha $x_n$ e $x_{n-1}$ . Salta para $x_{n-1}$ -ésima instrução se $x_n = true$
<b>pc</b>	empilha o endereço da instrução atual.
<b>halt</b>	Causa a parada da máquina de pilha.
Instruções de IO	
<b>getch</b>	Lê um carácter da entrada e o empilha.
<b>putch</b>	Converte o topo da pilha em carácter, e o imprime.

Tabela 2: Conjunto de instruções para manipulação da memória, fluxo de controle e IO

1. Para cada uma dos itens a seguir construa um programa equivalente na máquina de pilha, que preserve a semântica do programa original. Assuma que os programas estão escritos em C.

2<sup>1/2</sup>

```
int x = 10, y = 1;
int z = 2x*x - 4*x + 2;
boolean w = z == y;
```

2<sup>1/2</sup>

```
int i;
int n = 20;
int x[20];
x[0] = 1;
x[1] = 1;
for(i = 2; i < n; i++){
    x[i] = x[i-1] + x[i-2];
}
```

2<sup>1/2</sup> Neste item, lembre-se de preservar a semântica de chamada e retorno de função.

```
int foo(int n){
    int x = 2;
    while(n > 0){
        x = x * 2;
    }
    return n;
}
```

2<sup>1/2</sup> Neste item, lembre-se de preservar a semântica de chamada e retorno de função.

```
void foo(char * g, int n){
    int i;
    for(i = 0; i < n; i++){
        print(g[i]);
    }
}
```

2. (2pt points) (Extra) : Mostre como podemos definir, de maneira formal e genérica, a semântica da seguinte linguagem cuja sintaxe abstrata é:

```
CMD -> v <- E
      | 'while' E (CMD)*
E -> E + E
   | E * E
   | Int
```

3. (2pt points) (Extra) : Mostre como podemos definir, de maneira formal e genérica, a semântica de para chamada de função.
4. (3pt points) (Extra) Converta o seguinte programa em C para a máquina de pilha, preservando a semântica do programa original. Você deve preservar a semântica de chamada e retorno de função recursiva (Você deve dar a semântica de chamada de função de forma fidedigna.).

```
int fat(int n){
    if(n == 0) return 1;
    return n*fat(n-1);
}
void main(){
```

```
int x = 5;
fat(x);
print("ok1\n");
x = 9
fat(x);
print("ok2\n");
}
```